Project3Task0:

Task 0 Block.java

```java
/**

 Block class represents a single block in a blockchain.
 It contains the index, timestamp, data, previous hash, nonce, and difficulty
of the block.
 */
package edu.cmu.ds;
import com.google.gson.Gson;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

public class Block {
    // The hexadecimal characters used to convert byte arrays to strings
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
    // The index of the block in the blockchain
    private int index;

    // The timestamp of when the block was created
    private Timestamp timestamp;

    // The data stored in the block
    private String data;

    // The hash of the previous block in the blockchain
    public String previousHash;

    // The nonce used to find a hash that meets the block's difficulty level
    private BigInteger nonce;

    // The difficulty level of the block
    private int difficulty;

    /**
     * Constructor for the Block class.
     *
     * @param index      The index of the block in the blockchain
     * @param timestamp  The timestamp of when the block was created
     * @param data       The data stored in the block
     * @param difficulty The difficulty level of the block
     */
    Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
        this.nonce = BigInteger.ZERO;
    }
```

```java
    /**
     * Calculates the SHA-256 hash value of the block.
     *
     * @return The SHA-256 hash value of the block
     */
    public String calculateHash() {
        String information = String.format("%d%s%s%s%d%d", index,
timestamp.toString(), data, previousHash, nonce, difficulty);
        String hash_value = null;

        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(information.getBytes(StandardCharsets.UTF_8));
            hash_value = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException var4) {
            System.out.println("No Hash available" + var4);
        }

        return hash_value;
    }

    /**
     * Returns the data stored in the block.
     *
     * @return The data stored in the block
     */
    public String getData() {
        return data;
    }

    /**
     * Returns the difficulty level of the block.
     *
     * @return The difficulty level of the block
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Returns the index of the block in the blockchain.
     *
     * @return The index of the block in the blockchain
     */
    public int getIndex() {
        return index;
    }

    /**
     * Returns the nonce used to find a hash that meets the block's
difficulty level.
     *
     * @return The nonce used to find a hash that meets the block's
difficulty level
     */
    public BigInteger getNonce() {
        return nonce;
```

```
    }

    /**
     * Returns the hash of the previous block in the blockchain.
     *
     * @return The hash of the previous block in the blockchain
     */
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * Returns the timestamp of when the block was created.
     *
     * @return The timestamp of when the block was created
     */
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Calculates the proof-of-work for the current block.
     *
     * @return the hash value of the block after successful proof-of-work
     */
    public String proofOfWork() {
        String hash_value;

        while (true) {
            // Calculate the hash value
            hash_value = calculateHash();
            // Check if the hash value satisfies the difficulty
            if (hash_value.substring(0, difficulty).matches("0{" + difficulty
+ "}")) {
                break;
            }
            // Increment the nonce value and try again
            nonce = nonce.add(BigInteger.ONE);
        }

        return hash_value;
    }

    /**
     * Sets the data of the block.
     *
     * @param data the data to be set
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Sets the difficulty level for proof-of-work.
     *
     * @param difficulty the difficulty level to be set
     */
```

```
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Sets the index of the block.
     *
     * @param index the index to be set
     */
    public void setIndex(int index) {
        this.index = index;
    }

    /**
     * Sets the previous hash of the block.
     *
     * @param previousHash the previous hash to be set
     */
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Sets the timestamp of the block.
     *
     * @param timestamp the timestamp to be set
     */
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

    /**
     * Converts a byte array to a hexadecimal string.
     *
     * @param bytes the byte array to be converted
     * @return the hexadecimal string representation of the byte array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];

        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 255;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 15];
        }

        return new String(hexChars);
    }
}
```

Task 0 BlockChain.java

```
/**
```

```java
/* This class represents a simple implementation of a blockchain, allowing for
transactions to be added,
 verified, and repaired if necessary. It uses SHA-256 for hashing and proof
of work to ensure the integrity
 of the chain.
 */
package edu.cmu.ds;

import com.google.gson.Gson;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class BlockChain {
    // instance variables
    private List<Block> BlockChain = new ArrayList<>();
    private String chainHash = "";
    private int hashesPerSecond = 0;
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

    /**
     * Constructor for BlockChain class.
     */
    public BlockChain() {
    }

    /**
     * Main method for running the blockchain application.
     */
    public static void main(String[] args) {
        String[] menuOptions = new String[]{
                "View basic BlockChain status.",
                "Add a transaction to the BlockChain.",
                "Verify the BlockChain.",
                "View the BlockChain.",
                "Corrupt the chain.",
                "Hide the corruption by repairing the chain.",
                "Exit."
        };
        BlockChain bc = new BlockChain();
        Block genesisBlock = new Block(bc.getChainSize(), bc.getTime(),
"Genesis", 2);
        bc.addBlock(genesisBlock);
        bc.computeHashesPerSecond();
        Scanner inputScanner = new Scanner(System.in);
        boolean exit = false;

        while (!exit) {
            for (int i = 0; i < menuOptions.length; ++i) {
                System.out.println(i + ". " + menuOptions[i]);
            }
```

```
            int userOption = Integer.parseInt(inputScanner.nextLine());
            Timestamp startTime;
            Timestamp endTime;
            int timeElapsed;
            switch (userOption) {
                case 0:
                    displayBlockChainStatus(bc);
                    break;
                case 1:
                    addTransactionToBlockChain(inputScanner, bc);
                    break;
                case 2:
                    verifyBlockChain(bc);
                    break;
                case 3:
                    viewBlockChain(bc);
                    break;
                case 4:
                    corruptChain(inputScanner, bc);
                    break;
                case 5:
                    repairChain(bc);
                    break;
                case 6:
                    exit = true;
            }
        }

        inputScanner.close();
    }

    /**
     * Displays the current status of the blockchain.
     * @param bc BlockChain instance to display status for.
     */
    private static void displayBlockChainStatus(BlockChain bc) {
        System.out.println("Current size of chain: " + bc.getChainSize());
        System.out.println("Difficulty of most recent block: " +
bc.getLatestBlock().getDifficulty());
        System.out.println("Total difficulty for all blocks: " +
bc.getTotalDifficulty());
        System.out.println("Approximate hashes per second on this machine: "
+ bc.getHashespersecond());
        System.out.println("Expected total hashes required for the whole
chain: " + bc.getTotalExpectedHashes());
        System.out.println("Nonce for most recent block: " +
bc.getLatestBlock().getNonce());
        System.out.println("Chain hash: " + bc.getChainHash());
    }

    /**
     * Allows a user to add a transaction to the blockchain.
     * @param inputScanner Scanner instance to read user input.
     * @param bc BlockChain instance to add transaction to.
     */
    private static void addTransactionToBlockChain(Scanner inputScanner,
BlockChain bc) {
```

```
        System.out.println("Enter difficulty > 0");
        int difficulty = Integer.parseInt(inputScanner.nextLine());
        System.out.println("Enter transaction");
        String data = inputScanner.nextLine();
        Timestamp startTime = bc.getTime();
        bc.addBlock(new Block(bc.getChainSize(), bc.getTime(), data,
difficulty));
        Timestamp endTime = bc.getTime();
        int timeElapsed = (int) (endTime.getTime() - startTime.getTime());
        System.out.println("Total execution time to add this block was " +
timeElapsed + " milliseconds");
    }

    /**

     Verifies the integrity of the blockchain.
     @param bc The blockchain to verify.
     */
    private static void verifyBlockChain(BlockChain bc) {
        Timestamp startTime = bc.getTime();
        String validationResult = bc.isChainValid();
        Timestamp endTime = bc.getTime();
        int timeElapsed = (int) (endTime.getTime() - startTime.getTime());
        System.out.print("Chain verification: ");
        System.out.println(validationResult);
        System.out.println("Total execution time to verify the chain was " +
timeElapsed + " milliseconds");
    }
    /**

     Displays the blockchain.
     @param bc The blockchain to display.
     */
    private static void viewBlockChain(BlockChain bc) {
        System.out.println("View the BlockChain");
        System.out.println(bc.toString());
    }
    /**

     Corrupts a block in the blockchain.
     @param inputScanner The scanner to use for input.
     @param bc The blockchain to corrupt.
     */
    private static void corruptChain(Scanner inputScanner, BlockChain bc) {
        System.out.println("corrupt the BlockChain");
        System.out.println("Enter block ID of block to corrupt");
        int index = Integer.parseInt(inputScanner.nextLine());
        System.out.println("Enter new data for block " + index);
        String corruptMessage = inputScanner.nextLine();
        bc.getBlock(index).setData(corruptMessage);
        System.out.println("Block " + index + " now holds " +
corruptMessage);
    }
    /**

     Repairs the blockchain.
     @param bc The blockchain to repair.
```

```
     */
    private static void repairChain(BlockChain bc) {
        Timestamp startTime = bc.getTime();
        if (!bc.isChainValid().equals("True")) {
            bc.repairChain();
        }
        Timestamp endTime = bc.getTime();
        int timeElapsed = (int) (endTime.getTime() - startTime.getTime());
        System.out.println("Total execution time required to repair the chain
was " + timeElapsed + " milliseconds");
    }
    /**

     Gets the hash of the blockchain.
     @return The hash of the blockchain.
     */
    public String getChainHash() {
        return chainHash;
    }
    /**

     Gets the current time.
     @return The current time.
     */
    public Timestamp getTime() {
        return new Timestamp(System.currentTimeMillis());
    }
    /**

     Gets the latest block in the blockchain.
     @return The latest block in the blockchain.
     */
    public Block getLatestBlock() {
        return BlockChain.get(getChainSize() - 1);
    }
    /**
     Gets the size of the blockchain.
     @return The size of the blockchain.
     */
    public int getChainSize() {
        return BlockChain.size();
    }
    /**
     Gets the block at the specified index.
     @param i The index of the block to get.
     @return The block at the specified index.
     */
    public Block getBlock(int i) {
        if (i >= getChainSize()) {
            System.out.println("Insert number exceed block size");
            return null;
        } else {
            return BlockChain.get(i);
        }
    }

    /**
```

```
     * Calculates the number of hashes per second that can be computed by the
system.
     */
    public void computeHashesPerSecond() {
        String s = "0";
        Timestamp start = getTime();

        for (int i = 0; i < 1000000; ++i) {
            calculateHash(s);
        }

        Timestamp end = getTime();
        hashesPerSecond = (int) (1000000 / (double) (end.getTime() -
start.getTime()) * 1000.0);
    }

    /**
     * Returns the number of hashes per second that can be computed by the
system.
     */
    public int getHashespersecond() {
        return hashesPerSecond;
    }

    /**
     * Adds a block to the chain.
     * If the chain is empty, sets the previous hash to an empty string.
     * Otherwise, sets the previous hash to the hash of the previous block in
the chain.
     * Updates the chain hash to the proof of work of the new block.
     */
    public void addBlock(Block block) {
        if (getChainSize() == 0) {
            block.setPreviousHash("");
        } else {
            block.setPreviousHash(chainHash);
        }

        BlockChain.add(block);
        chainHash = block.proofOfWork();
    }

    /**
     * Converts the blockchain to a JSON string.
     */
    public String toString() {
        BlockChain bc = new BlockChain();

        for (int i = 0; i < getChainSize(); ++i) {
            bc.BlockChain.add(getBlock(i));
        }

        bc.hashesPerSecond = getHashespersecond();
        bc.chainHash = getChainHash();
        Gson gson = new Gson();
        String messageToSend = gson.toJson(bc);
        return messageToSend;
```

```
    }

    /**
     * Returns the total difficulty of the chain.
     */
    public int getTotalDifficulty() {
        int totalDifficulty = 0;

        for (Block b : BlockChain) {
            totalDifficulty += b.getDifficulty();
        }

        return totalDifficulty;
    }

    /**
     * Returns the total expected hashes of the chain.
     */
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0.0;

        for (Block b : BlockChain) {
            totalExpectedHashes += Math.pow(16.0, (double)
b.getDifficulty());
        }

        return totalExpectedHashes;
    }

    /**
     * Checks the validity of the chain.
     * Returns "True" if the chain is valid, otherwise returns an error
message.
     */
    public String isChainValid() {
        for (int i = 0; i < getChainSize(); ++i) {
            Block b = getBlock(i);
            String s = b.calculateHash();

            if (!hashIsValid(s, b.getDifficulty())) {
                return "FALSE\nImproper hash on node " + i + " does not begin
with " + "00";
            }

            if (i != 0 && !previousHashIsValid(i, b)) {
                return "Block " + i + " does not have a matching hash.";
            }
        }

        if (!lastBlockHashIsValid()) {
            return "The chain hash is different from the hash of the last
block.";
        } else {
            return "True";
        }
    }
```

```
    /**
     * This method checks if a hash is valid for a given difficulty level.
     *
     * @param hash       the hash to check.
     * @param difficulty the difficulty level to check against.
     * @return true if the hash is valid for the given difficulty level,
false otherwise.
     */
    private boolean hashIsValid(String hash, int difficulty) {
        for (int j = 0; j < difficulty; ++j) {
            if (hash.charAt(j) != '0') {
                return false;
            }
        }
        return true;
    }

    /**
     * This method checks if the previous block's hash is valid for a given
block.
     *
     * @param index the index of the previous block.
     * @param block the current block to check against.
     * @return true if the previous block's hash is valid for the given
block, false otherwise.
     */
    private boolean previousHashIsValid(int index, Block block) {
        return getBlock(index -
1).calculateHash().equals(block.getPreviousHash());
    }

    /**
     * This method checks if the last block's hash is valid for the current
chain.
     *
     * @return true if the last block's hash is valid for the current chain,
false otherwise.
     */
    private boolean lastBlockHashIsValid() {
        return getBlock(getChainSize() -
1).calculateHash().equals(chainHash);
    }

    /**
     * This method repairs the chain by updating any invalid hashes.
     */
    public void repairChain() {
        for (int i = 0; i < getChainSize(); ++i) {
            Block b = getBlock(i);
            updateChainHashes(i, b);
        }
    }

    /**
     * This method updates the hashes for the chain.
     *
     * @param index the index of the block to update.
```

```
     * @param block the block to update.
     */
    private void updateChainHashes(int index, Block block) {
        if (index != getChainSize() - 1) {
            getBlock(index + 1).previousHash = block.proofOfWork();
        } else {
            chainHash = block.proofOfWork();
        }
    }

    /**
     Calculates the SHA-256 hash of the given string.
     @param s the string to be hashed
     @return the hash value as a hexadecimal string
     */
    public String calculateHash(String s) {
        String hashValue = null;
        try {
            // Create a new SHA-256 message digest instance
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            // Update the digest with the UTF-8 encoded bytes of the input
string
            md.update(s.getBytes(StandardCharsets.UTF_8));
            // Convert the digest bytes to a hexadecimal string
            hashValue = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException var4) {
            // If SHA-256 is not available, print an error message and return
null
            System.out.println("SHA-256 hash algorithm is not available: " +
var4);
        }
        return String.valueOf(hashValue);
    }

    /**
     Converts the given byte array to a hexadecimal string.
     @param bytes the byte array to be converted
     @return the hexadecimal string
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; ++j) {
            int v = bytes[j] & 255;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 15];
        }
        return new String(hexChars);
    }
}
```

Task 0 Execution

0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2202643
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 116
Chain hash: 001F760C3F7913DD0354C3EC8F9447C72FBD7B66B9BF5A2651F0660F8D8BF05C
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Alice pays Bill 100 DSCoin
Total execution time to add this block was 3 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Bill pays Clara 50 DSCoin
Total execution time to add this block was 5 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.

3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Clara pays Daisy 10 DS Coin
Total execution time to add this block was 10 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: True
Total execution time to verify the chain was 1 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
3
View the BlockChain
{"BlockChain":[{"index":0,"timestamp":"Mar 18, 2023, 10:48:40 PM","data":"Genesis","previousHash":"","nonce":116,"difficulty":2},{"index":1,"timestamp":"Mar 18, 2023, 10:49:17 PM","data":"Alice pays Bill 100 DSCoin","previousHash":"001F760C3F7913DD0354C3EC8F9447C72FBD7B66B9BF5A2651F0660F8D8BF05C","nonce":38,"difficulty":2},{"index":2,"timestamp":"Mar 18, 2023, 10:49:33 PM","data":"Bill pays Clara 50 DSCoin","previousHash":"00E00192013FF19134EFB04C25B70D92CC006E7EF04D8AE4645F43CD55951442","nonce":95,"difficulty":2},{"index":3,"timestamp":"Mar 18, 2023, 10:49:48 PM","data":"Clara pays Daisy 10 DS Coin","previousHash":"002F8D7977591D02D437626019A9C6A54279B73E485D49B615810FD268307E85","nonce":352,"difficulty":2}],"chainHash":"00775448B8BBB35345EA7EC0A59D1A4631FDD6EAF2C3CAAAB365D50E8554119B","hashesPerSecond":2202643}
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.

3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
4
corrupt the BlockChain
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bill 76 DSCoin
Block 1 now holds Alice pays Bill 76 DSCoin
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
3
View the BlockChain
{"BlockChain":[{"index":0,"timestamp":"Mar 18, 2023, 10:48:40
PM","data":"Genesis","previousHash":"","nonce":116,"difficulty":2},{"index":1,"timestamp":"M
ar 18, 2023, 10:49:17 PM","data":"Alice pays Bill 76
DSCoin","previousHash":"001F760C3F7913DD0354C3EC8F9447C72FBD7B66B9BF5A2651F0660
F8D8BF05C","nonce":38,"difficulty":2},{"index":2,"timestamp":"Mar 18, 2023, 10:49:33
PM","data":"Bill pays Clara 50
DSCoin","previousHash":"00E00192013FF19134EFB04C25B70D92CC006E7EF04D8AE4645F43C
D55951442","nonce":95,"difficulty":2},{"index":3,"timestamp":"Mar 18, 2023, 10:49:48
PM","data":"Clara pays Daisy 10 DS
Coin","previousHash":"002F8D7977591D02D437626019A9C6A54279B73E485D49B615810FD2
68307E85","nonce":352,"difficulty":2}],"chainHash":"00775448B8BBB35345EA7EC0A59D1A463
1FDD6EAF2C3CAAAB365D50E8554119B","hashesPerSecond":2202643}
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: FALSE
Improper hash on node 1 does not begin with 00
Total execution time to verify the chain was 0 milliseconds
0. View basic BlockChain status.

1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
5
Total execution time required to repair the chain was 13 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: True
Total execution time to verify the chain was 0 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
4
Enter transaction
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 515 milliseconds
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
0
Current size of chain: 5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine: 2202643
Expected total hashes required for the whole chain: 66560.0

Nonce for most recent block: 264138
Chain hash: 00008C698BD93569DF48A5B3EA5452B292614A501E83CB005C5CF1A8E90505D8
0. View basic BlockChain status.
1. Add a transaction to the BlockChain.
2. Verify the BlockChain.
3. View the BlockChain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
6

Process finished with exit code 0

Project3Task1:

Task 1 Client Side Execution

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2331002
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 251
Chain hash: 00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10815575334
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2

Enter transaction
Alice pays Bill 100 DSCoin
Total execution time to add this block was 12 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Bill pays Clara 50 DSCoin
Total execution time to add this block was 7 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Clara pays Daisy 10 DS Coin
Total execution time to add this block was 1 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: True
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.

5. Hide the corruption by repairing the chain.
6. Exit.
3
View the Blockchain
{"blockchain":[{"index":0,"timestamp":"Mar 19, 2023, 5:01:26
PM","data":"Genesis","previousHash":"","nonce":251,"difficulty":2},{"index":1,"timestamp":"M
ar 19, 2023, 5:01:41 PM","data":"Alice pays Bill 100
DSCoin","previousHash":"00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10
815575334","nonce":559,"difficulty":2},{"index":2,"timestamp":"Mar 19, 2023, 5:01:47
PM","data":"Bill pays Clara 50
DSCoin","previousHash":"003AF86978FC4B0973ECFB6F43C288186701A8E8485ED01F09506A6
F1C6A0DF9","nonce":696,"difficulty":2},{"index":3,"timestamp":"Mar 19, 2023, 5:01:53
PM","data":"Clara pays Daisy 10 DS
Coin","previousHash":"00BE26FD6DF862EA35B68776989F0AEABD2D252231EFC3EF1A37DF1D
E31416D8","nonce":63,"difficulty":2}],"chainHash":"00B25ECD248A83436E771665B4950B464B
B1AF081FF2A393B54BB144B6B1998A","hashesPerSecond":2331002}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
4
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bill 76 DSCoin
Block 1 now holds Alice pays Bill 76 DSCoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
3
View the Blockchain
{"blockchain":[{"index":0,"timestamp":"Mar 19, 2023, 5:01:26
PM","data":"Genesis","previousHash":"","nonce":251,"difficulty":2},{"index":1,"timestamp":"M
ar 19, 2023, 5:01:41 PM","data":"Alice pays Bill 76
DSCoin","previousHash":"00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10
815575334","nonce":559,"difficulty":2},{"index":2,"timestamp":"Mar 19, 2023, 5:01:47
PM","data":"Bill pays Clara 50

DSCoin","previousHash":"003AF86978FC4B0973ECFB6F43C288186701A8E8485ED01F09506A6F1C6A0DF9","nonce":696,"difficulty":2},{"index":3,"timestamp":"Mar 19, 2023, 5:01:53 PM","data":"Clara pays Daisy 10 DS Coin","previousHash":"00BE26FD6DF862EA35B68776989F0AEABD2D252231EFC3EF1A37DF1DE31416D8","nonce":63,"difficulty":2}],"chainHash":"00B25ECD248A83436E771665B4950B464BB1AF081FF2A393B54BB144B6B1998A","hashesPerSecond":2331002}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: False
Improper hash on node 1 does not begin with 00
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
5
Total execution time required to repair the chain was 3 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
2
Chain verification: True
Total execution time required to verify the chain was 1 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
1

Enter difficulty > 0
4
Enter transaction
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 73 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
0
Current size of chain: 5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine: 2331002
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 25510
Chain hash: 00004A0DCA5B38410E7CE4277224CC02FBCD28722A06344705C529E982EB8793
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
6

Process finished with exit code 0

Task 1 Server Side Execution

Blockchain server running
We have a visitor
Response :
{"selection":0,"size":1,"chainHash":"00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10815575334","totalHashes":256.0,"totalDiff":2,"recentNonce":251,"diff":2,"hps":2331002}
Adding a block
Setting response to Total execution time to add this block was 12 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 12 milliseconds"}
Adding a block

Setting response to Total execution time to add this block was 7 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 7 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 1 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 1 milliseconds"}
Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 0 milliseconds
Setting response to Total execution time required to verify the chain was 0 milliseconds
View the Blockchain
Setting response to {"blockchain":[{"index":0,"timestamp":"Mar 19, 2023, 5:01:26 PM","data":"Genesis","previousHash":"","nonce":251,"difficulty":2},{"index":1,"timestamp":"Mar 19, 2023, 5:01:41 PM","data":"Alice pays Bill 100 DSCoin","previousHash":"00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10815575334","nonce":559,"difficulty":2},{"index":2,"timestamp":"Mar 19, 2023, 5:01:47 PM","data":"Bill pays Clara 50 DSCoin","previousHash":"003AF86978FC4B0973ECFB6F43C288186701A8E8485ED01F09506A6F1C6A0DF9","nonce":696,"difficulty":2},{"index":3,"timestamp":"Mar 19, 2023, 5:01:53 PM","data":"Clara pays Daisy 10 DS Coin","previousHash":"00BE26FD6DF862EA35B68776989F0AEABD2D252231EFC3EF1A37DF1DE31416D8","nonce":63,"difficulty":2}],"chainHash":"00B25ECD248A83436E771665B4950B464BB1AF081FF2A393B54BB144B6B1998A","hashesPerSecond":2331002}
Corrupt the Blockchain
Block 1 now holds Alice pays Bill 76 DSCoin
View the Blockchain
Setting response to {"blockchain":[{"index":0,"timestamp":"Mar 19, 2023, 5:01:26 PM","data":"Genesis","previousHash":"","nonce":251,"difficulty":2},{"index":1,"timestamp":"Mar 19, 2023, 5:01:41 PM","data":"Alice pays Bill 76 DSCoin","previousHash":"00615660F82E4BE83E648C72395A17448498D3940BDE72CC670BF10815575334","nonce":559,"difficulty":2},{"index":2,"timestamp":"Mar 19, 2023, 5:01:47 PM","data":"Bill pays Clara 50 DSCoin","previousHash":"003AF86978FC4B0973ECFB6F43C288186701A8E8485ED01F09506A6F1C6A0DF9","nonce":696,"difficulty":2},{"index":3,"timestamp":"Mar 19, 2023, 5:01:53 PM","data":"Clara pays Daisy 10 DS Coin","previousHash":"00BE26FD6DF862EA35B68776989F0AEABD2D252231EFC3EF1A37DF1DE31416D8","nonce":63,"difficulty":2}],"chainHash":"00B25ECD248A83436E771665B4950B464BB1AF081FF2A393B54BB144B6B1998A","hashesPerSecond":2331002}
Verifying entire chain
Chain verification: False
Improper hash on node 1 does not begin with 00
Total execution time required to verify the chain was 0 milliseconds
Setting response to Total execution time required to verify the chain was 0 milliseconds
Repairing the entire chain
Setting response to Total execution time required to repair the chain was 3 milliseconds

Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 1 milliseconds
Setting response to Total execution time required to verify the chain was 1 milliseconds
Adding a block
Setting response to Total execution time to add this block was 73 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 73 milliseconds"}
Response :
{"selection":0,"size":5,"chainHash":"00004A0DCA5B38410E7CE4277224CC02FBCD28722A06344705C529E982EB8793","totalHashes":66560.0,"totalDiff":12,"recentNonce":25510,"diff":4,"hps":2331002}

Task 1 RequestMessage.java

```
package edu.cmu.ds;
import com.google.gson.Gson;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

import java.net.*;
import java.io.*;
import java.util.Scanner;

/**
 This class is responsible for sending requests to the blockchain server and
printing responses to the console.
 */
public class RequestMessage {
    // Declare necessary variables
    static Socket clientSocket;
    static int serverPort = 7777;
    static BufferedReader in;
    static PrintWriter out;
    static Scanner readInput = new Scanner(System.in);
    static boolean finish = false;
    static JSONObject json = new JSONObject();

    /**
     This method initializes a connection with the blockchain server and
prompts the user to enter a selection until
     they choose to exit.
     @param args command line arguments
     */
    public static void main(String[] args) {
        try {
            // Connect to the server
            clientSocket = new Socket("localhost", serverPort);
            BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
            do {
                // Prompt the user for a selection and send it to the server
                int option = getSelection();
```

```
                if (finish) break;
                pass(option);
            } while (true);
        } catch (IOException e) {
            System.out.println("IO Exception: " + e.getMessage());
        } finally {
            // Close the connection with the server
            try {
                if (clientSocket != null) clientSocket.close();
            } catch (IOException e) {
                // Ignored
            }
        }
    }

    /**
     This method prompts the user for a selection and returns the integer
value of the selection.
     @return an integer representing the user's selection
     */
    public static int getSelection() {
        // Declare an array of messages to display to the user for each
selection option
        String[] message = {"View basic blockchain status.", "Add a
transaction to the blockchain.",
                "Verify the blockchain.", "View the blockchain.", "Corrupt
the chain.",
                "Hide the corruption by repairing the chain.", "Exit."};
        // Clear the JSON object
        json.clear();

        // Display the selection options to the user
        for (int i = 0; i < message.length; i++) {
            System.out.println(i + ". " + message[i]);
        }

        // Read the user's selection and add the appropriate data to the JSON
object
        int option = Integer.parseInt(readInput.nextLine());
        switch (option) {
            case 0:
                addSelectionToJSON(option);
                break;
            case 1:
                addTransactionDataToJSON(option);
                break;
            case 2:
            case 3:
                addSelectionToJSON(option);
                break;
            case 4:
                addCorruptDataToJSON(option);
                break;
            case 5:
                addSelectionToJSON(option);
                break;
            case 6:
```

```
                finish = true;
                break;
            default:
                break;
        }
        return option;
    }

    /**
     This method sends the user's selection to the server and waits for a
response. It then handles the response
     appropriately based on the user's original selection.
     @param option an integer representing the user's selection
     */
    public static void pass(int option) {
        try {
            // Initialize input and output streams
            in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            // Send the JSON object to the server
            out.println(json.toJSONString());
            out.flush();

            // Receive the server's response as a JSON object
            json = (JSONObject) JSONValue.parse(in.readLine());

            // Handle the server's response based on the user's selection
            handleServerResponse(option);
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
    }

    /**

     This method adds the user's selection to the JSON object.
     @param option an integer representing the user's selection
     */
    private static void addSelectionToJSON(int option) {
        json.put("selection", option);
    }

    /**
     This method adds the user's transaction data to the JSON object.
     @param option an integer representing the user's selection
     */
    private static void addTransactionDataToJSON(int option) {
        // Prompt the user for the transaction data and difficulty
        System.out.println("Enter difficulty > 0");
        int difficulty = Integer.parseInt(readInput.nextLine());
        System.out.println("Enter transaction");
        String data = readInput.nextLine();

        // Add the transaction data and difficulty to the JSON object
        json.put("selection", option);
```

```
        json.put("difficulty", difficulty);
        json.put("data", data);
    }

    /**
     This method adds the user's corrupt data to the JSON object.
     @param option an integer representing the user's selection
     */
    private static void addCorruptDataToJSON(int option) {
        // Prompt the user for the block index and new corrupt data
        System.out.println("Enter block ID of block to corrupt");
        int index = Integer.parseInt(readInput.nextLine());
        System.out.println("Enter new data for block " + index);
        String corruptMessage = readInput.nextLine();

        // Add the block index and new corrupt data to the JSON object
        json.put("selection", option);
        json.put("index", index);
        json.put("data", corruptMessage);
    }

    /**
     This method handles the server's response based on the user's selection.
     @param option an integer representing the user's selection
     */
    private static void handleServerResponse(int option) {
        switch (option) {
            case 0:
                printBlockchainStatus();
                break;
            case 1:
            case 4:
            case 5:
                printResponseMessage();
                break;
            case 2:
                printVerificationResult();
                break;
            case 3:
                printBlockchain();
                break;
        }
    }

    /**
     This method prints the blockchain status to the console.
     */
    private static void printBlockchainStatus() {
        System.out.println("Current size of chain: " + ((Long)
json.get("size")).intValue());
        System.out.println("Difficulty of most recent block: " + ((Long)
json.get("diff")).intValue());
        System.out.println("Total difficulty for all blocks: " + ((Long)
json.get("totalDiff")).intValue());
        System.out.println("Approximate hashes per second on this machine: "
+ ((Long) json.get("hps")).intValue());
        System.out.println("Expected total hashes required for the whole
```

```
chain: " + (double) json.get("totalHashes"));
        System.out.println("Nonce for most recent block: " + ((Long)
json.get("recentNonce")).intValue());
        System.out.println("Chain hash: " + json.get("chainHash"));
    }

    /**
     This method prints the server's response message to the console.
     */
    private static void printResponseMessage() {
        System.out.println(json.get("response"));
    }

    /**
     This method prints the verification result to the console.
     */
    private static void printVerificationResult() {
        System.out.println("Chain verification: " +
json.get("verification"));
        if (json.get("verification").equals("False")) {
            System.out.println(json.get("errorMessage"));
        }
        printResponseMessage();
    }

    /**
     This method prints the blockchain to the console.
     */
    private static void printBlockchain() {
        System.out.println("View the Blockchain");
        printResponseMessage();
    }
}
```

Task 1 ResponseMessage.java

```
package edu.cmu.ds;

import com.google.gson.Gson;
import java.net.*;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
/**
 A class to handle responses to client requests in a blockchain server
 */
```

```
public class ResponseMessage {
    // Static variables to be shared across instances of ResponseMessage
    static Socket clientSocket = null;
    static int serverPort = 7777;
    static JSONObject listenJson = new JSONObject();
    static JSONObject returnJson = new JSONObject();
    static BlockChain bc;
    /**
     The main method to start the blockchain server and handle client
requests
     @param args Command line arguments (not used)
     */
    public static void main(String[] args) {
        initializeBlockChain();
        System.out.println("Blockchain server running");
        try (ServerSocket listenSocket = new ServerSocket(serverPort)) {
            while (true) {
                clientSocket = listenSocket.accept();
                handleClientRequest();
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            closeClientSocket();
        }
    }
    /**
     A helper method to initialize the blockchain
     */
    private static void initializeBlockChain() {
        bc = new BlockChain();
        Block b = new Block(bc.getChainSize(), bc.getCurrentTime(),
"Genesis", 2);
        bc.insertBlock(b);
        bc.calculateHashesPerSecond();
    }

    /**
     A helper method to handle a client request
     */
    private static void handleClientRequest() {
        try (Scanner in = new Scanner(clientSocket.getInputStream());
             PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())))) {

            System.out.println("We have a visitor");
            // Continuously listen for client messages and respond to them
            while (true) {
                if (in.hasNextLine()) {
                    String info = in.nextLine();
                    listenJson = (JSONObject) JSONValue.parse(info);
                    Long l = (Long) listenJson.get("selection");
                    int option = l.intValue();
                    process(option);
                    out.println(returnJson.toJSONString());
                    out.flush();
                } else {
```

```
                    break;
                }
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
    }
    /**
     A helper method to close the client socket
     */
    private static void closeClientSocket() {
        try {
            if (clientSocket != null) clientSocket.close();
        } catch (IOException e) {
            // Ignored
        }
    }

    /**
     This method processes the selected option and updates the returnJson
object with the appropriate response.
     @param option an integer representing the selected option
     */
    public static void process(int option) {
        // Clear the returnJson object to start with a fresh response
        returnJson.clear();

        // Declare variables to hold start and end times for timing execution
and result messages
        long startTime, endTime;
        String resultMessage;

        // Process the selected option based on its integer value
        switch (option) {
            case 0:
                // If option 0 is selected, process the case for option 0
                processCaseZero();
                break;
            case 1:
                // If option 1 is selected, insert a new block into the
blockchain and time the execution
                System.out.println("Adding a block");
                startTime = System.currentTimeMillis();
                bc.insertBlock(new Block(bc.getChainSize(),
bc.getCurrentTime(), (String) listenJson.get("data"), ((Long)
listenJson.get("difficulty")).intValue()));
                endTime = System.currentTimeMillis();
                // Calculate the total execution time and add a message to
the response object
                resultMessage = "Total execution time to add this block was "
+ (endTime - startTime) + " milliseconds";
                updateReturnJson(1, resultMessage);
                break;
            case 2:
                // If option 2 is selected, validate the entire blockchain
and time the execution
                System.out.println("Verifying entire chain");
```

```
                startTime = System.currentTimeMillis();
                String validation = bc.validateChain();
                endTime = System.currentTimeMillis();
                // Calculate the total execution time and add a message and
validation result to the response object
                resultMessage = "Total execution time required to verify the
chain was " + (endTime - startTime) + " milliseconds";
                updateReturnJson(2, validation, resultMessage);
                break;
            case 3:
                // If option 3 is selected, update the response object with
the current state of the blockchain
                updateReturnJson(bc.toString());
                break;
            case 4:
                // If option 4 is selected, corrupt the data of a block at a
specified index and add a message to the response object
                int index = ((Long) listenJson.get("index")).intValue();
                String corruptMessage = (String) listenJson.get("data");
                processCaseFour(index, corruptMessage);
                break;
            case 5:
                // If option 5 is selected, repair the entire blockchain and
time the execution
                processCaseFive();
                break;
        }
    }

    /**
     This method updates the returnJson object with the appropriate response
for option 0.
     */
    private static void processCaseZero() {
        // Add the necessary data to the response object
        returnJson.put("selection", 0);
        returnJson.put("size", bc.getChainSize());
        returnJson.put("chainHash", bc.getChainHash());
        returnJson.put("totalHashes", bc.calculateTotalExpectedHashes());
        returnJson.put("totalDiff", bc.computeTotalDifficulty());
        returnJson.put("recentNonce", bc.getBlock(bc.getChainSize() -
1).getNonce());
        returnJson.put("diff", bc.getBlock(bc.getChainSize() -
1).getDifficulty());
        returnJson.put("hps", bc.getHashRate());
        // Print the response object to the console
        System.out.println("Response : " + returnJson.toJSONString());
    }

    /**
     This method corrupts the data of a block at a specified index and
updates the returnJson object with a response message.
     @param index an integer representing the index of the block to corrupt
     @param corruptMessage a string containing the new, corrupted data to add
to the block
     */
    private static void processCaseFour(int index, String corruptMessage) {
```

```
        System.out.println("Corrupt the Blockchain");
        // Set the data of the specified block to the new, corrupted data
        bc.getBlock(index).setData(corruptMessage);
        // Create a response message with information about the corrupted
block and add it to the response object
        String resultMessage = "Block " + index + " now holds " +
corruptMessage;
        System.out.println(resultMessage);
        returnJson.put("response", resultMessage);
    }

    /**
     This method repairs the entire blockchain and updates the returnJson
object with a response message.
     */
    private static void processCaseFive() {
        System.out.println("Repairing the entire chain");
        long startTime = System.currentTimeMillis();
        // Validate the chain and, if it is not valid, repair it
        if (!bc.validateChain().equals("True")) {
            bc.fixChain();
        }
        long endTime = System.currentTimeMillis();
        // Calculate the total execution time and add a message to the
response object
        String resultMessage = "Total execution time required to repair the
chain was " + (endTime - startTime) + " milliseconds";
        System.out.println("Setting response to " + resultMessage);
        returnJson.put("response", resultMessage);
    }

    /**
     This method updates the returnJson object with the selected option and a
result message.
     @param selection an integer representing the selected option
     @param resultMessage a string containing the result message
     */
    private static void updateReturnJson(int selection, String resultMessage)
{
        returnJson.put("selection", selection);
        returnJson.put("response", resultMessage);
        System.out.println("Setting response to " + resultMessage);
        System.out.println("..." + returnJson.toJSONString());
    }

    /**
     This method updates the returnJson object with the current state of the
blockchain.
     @param resultMessage a string containing the current state of the
blockchain
     */
    private static void updateReturnJson(String resultMessage) {
        System.out.println("View the Blockchain");
        System.out.println("Setting response to " + resultMessage);
        returnJson.put("response", resultMessage);
    }
```

```java
    /**
     This method updates the returnJson object with the selected option, a
validation result, and a result message.
     @param selection an integer representing the selected option
     @param validation a string containing the validation result of the
blockchain
     @param resultMessage a string containing the result message
     */
    private static void updateReturnJson(int selection, String validation,
String resultMessage) {
        returnJson.put("selection", selection);
        // If the validation result is true, add "True" to the response
object, otherwise add "False" and an error message
        returnJson.put("verification", validation.equals("True") ? "True" :
"False");
        if (!validation.equals("True")) {
            returnJson.put("errorMessage", validation);
            System.out.println("Chain verification: False");
            System.out.println(validation);
        } else {
            System.out.println("Chain verification: TRUE");
        }
        returnJson.put("response", resultMessage);
        System.out.println(resultMessage);
        System.out.println("Setting response to " + resultMessage);
    }

    public static class BlockChain extends java.lang.Object{
        // instance variables
        private List<Block> blockchain;
        private String chainHash;
        private int hashesPerSecond;
        private static final char[] HEX_ARRAY =
"0123456789ABCDEF".toCharArray();

        /**
         * Constructor for BlockChain class.
         */
        public BlockChain() {
            blockchain = new ArrayList<>();
            chainHash = "";
            hashesPerSecond = 0;
        }
        /**
         Gets the hash of the blockchain.
         @return The hash of the blockchain.
         */
        public String getChainHash() {
            return chainHash;
        }

        /**
         Gets the current time.
         @return The current time.
         */
        public Timestamp getCurrentTime() {
            return new Timestamp(System.currentTimeMillis());
```

```
        }

        /**
         Gets the latest block in the blockchain.
         @return The latest block in the blockchain.
         */
        public Block getLatestBlock() {
            return blockchain.get(this.getChainSize() - 1);
        }

        /**
         Gets the size of the blockchain.
         @return The size of the blockchain.
         */
        public int getChainSize() {
            return blockchain.size();
        }

        /**
         Gets the block at the specified index.
         @param i The index of the block to get.
         @return The block at the specified index.
         */
        public Block getBlock(int i) {
            if (i >= getChainSize()) {
                System.out.println("Insert number exceed block size");
                return null;
            }
            return blockchain.get(i);
        }

        /**
         * Calculates the number of hashes per second that can be computed by
the system.
         */
        public void calculateHashesPerSecond() {
            String s = "0";
            Timestamp start = getCurrentTime();
            for (int i = 0; i < 1000000; i++) {
                computeHash(s);
            }
            Timestamp end = getCurrentTime();
            hashesPerSecond = (int) ((double) 1000000 / (end.getTime() -
start.getTime()) * 1000);
        }

        /**
         * Returns the number of hashes per second that can be computed by
the system.
         */
        public int getHashRate() {
            return hashesPerSecond;
        }

        /**
         * Adds a block to the chain.
         * If the chain is empty, sets the previous hash to an empty string.
```

```
         * Otherwise, sets the previous hash to the hash of the previous
block in the chain.
         * Updates the chain hash to the proof of work of the new block.
         */
        public void insertBlock(Block block) {
            if (getChainSize() == 0) {
                block.setPreviousHash("");
            } else {
                block.setPreviousHash(chainHash);
            }
            blockchain.add(block);
            chainHash = block.proofOfWork();
        }

        /**
         * Converts the blockchain to a JSON string.
         */
        @Override
        public String toString() {
            BlockChain tempChain = new BlockChain();
            for (int i = 0; i < getChainSize(); i++) {
                tempChain.blockchain.add(getBlock(i));
            }
            tempChain.hashesPerSecond = getHashRate();
            tempChain.chainHash = getChainHash();
            Gson gson = new Gson();
            return gson.toJson(tempChain);
        }

        /**
         * Returns the total difficulty of the chain.
         */
        public int computeTotalDifficulty() {
            int totalDifficulty = 0;
            for (Block b : blockchain) {
                totalDifficulty += b.getDifficulty();
            }
            return totalDifficulty;
        }

        /**
         * Returns the total expected hashes of the chain.
         */
        public double calculateTotalExpectedHashes() {
            double totalExpectedHashes = 0;
            for (Block b : blockchain) {
                totalExpectedHashes += Math.pow(16, b.getDifficulty());
            }
            return totalExpectedHashes;
        }

        /**
         * Checks the validity of the chain.
         * Returns "True" if the chain is valid, otherwise returns an error
message.
         */
        public String validateChain() {
```

```
            for (int i = 0; i < getChainSize(); i++) {
                Block b = getBlock(i);
                String s = b.calculateHash();
                for (int j = 0; j < b.getDifficulty(); j++) {
                    if (s.charAt(j) != '0') {
                        return "Improper hash on node " + i + " does not
begin with " +
                                "00";
                    }
                }
                if (i != 0 && !getBlock(i -
1).calculateHash().equals(b.getPreviousHash())) {
                    return "Block " + i + " does not have a matching hash.";
                }
            }
            if (!getBlock(getChainSize() -
1).calculateHash().equals(chainHash)) {
                return "The chain hash is different from the hash of the last
block.";
            }
            return "True";
        }

        /**
         * This method repairs the chain by updating any invalid hashes.
         */
        public void fixChain() {
            for (int i = 0; i < getChainSize(); i++) {
                Block b = getBlock(i);
                if (i != getChainSize() - 1) {
                    getBlock(i + 1).previousHash = b.proofOfWork();
                } else {
                    chainHash = b.proofOfWork();
                }
            }
        }

        /**
         Calculates the SHA-256 hash of the given string.
         @param s the string to be hashed
         @return the hash value as a hexadecimal string
         */
        public String computeHash(String s) {
            String hashValue = null;
            try {
                MessageDigest md;
                md = MessageDigest.getInstance("SHA-256");
                md.update(s.getBytes(StandardCharsets.UTF_8));
                hashValue = byteArrayToHex(md.digest());
            } catch (NoSuchAlgorithmException e) {
                System.out.println("No Hash available" + e);
            }
            return String.valueOf(hashValue);
        }
        /**
         Converts the given byte array to a hexadecimal string.
         @param bytes the byte array to be converted
```

```
         @return the hexadecimal string
         */
        public static String byteArrayToHex(byte[] bytes) {
            char[] hexChars = new char[bytes.length * 2];
            for (int j = 0; j < bytes.length; j++) {
                int v = bytes[j] & 0xFF;
                hexChars[j * 2] = HEX_ARRAY[v >>> 4];
                hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
            }
            return new String(hexChars);
        }
    }
}
```

Task 1 Block.java

```
/**

 Block class represents a single block in a blockchain.
 It contains the index, timestamp, data, previous hash, nonce, and difficulty
of the block.
 */
package edu.cmu.ds;
import com.google.gson.Gson;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

public class Block {
    // The hexadecimal characters used to convert byte arrays to strings
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
    // The index of the block in the blockchain
    private int index;

    // The timestamp of when the block was created
    private Timestamp timestamp;

    // The data stored in the block
    private String data;

    // The hash of the previous block in the blockchain
    public String previousHash;

    // The nonce used to find a hash that meets the block's difficulty level
    private BigInteger nonce;

    // The difficulty level of the block
    private int difficulty;

    /**
     * Constructor for the Block class.
     *
     * @param index      The index of the block in the blockchain
```

```
     * @param timestamp  The timestamp of when the block was created
     * @param data       The data stored in the block
     * @param difficulty The difficulty level of the block
     */
    Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
        this.nonce = BigInteger.ZERO;
    }

    /**
     * Calculates the SHA-256 hash value of the block.
     *
     * @return The SHA-256 hash value of the block
     */
    public String calculateHash() {
        String information = String.format("%d%s%s%s%d%d", index,
timestamp.toString(), data, previousHash, nonce, difficulty);
        String hash_value = null;

        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(information.getBytes(StandardCharsets.UTF_8));
            hash_value = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException var4) {
            System.out.println("No Hash available" + var4);
        }

        return hash_value;
    }

    /**
     * Returns the data stored in the block.
     *
     * @return The data stored in the block
     */
    public String getData() {
        return data;
    }

    /**
     * Returns the difficulty level of the block.
     *
     * @return The difficulty level of the block
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Returns the index of the block in the blockchain.
     *
     * @return The index of the block in the blockchain
     */
    public int getIndex() {
```

```
        return index;
    }

    /**
     * Returns the nonce used to find a hash that meets the block's
difficulty level.
     *
     * @return The nonce used to find a hash that meets the block's
difficulty level
     */
    public BigInteger getNonce() {
        return nonce;
    }

    /**
     * Returns the hash of the previous block in the blockchain.
     *
     * @return The hash of the previous block in the blockchain
     */
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * Returns the timestamp of when the block was created.
     *
     * @return The timestamp of when the block was created
     */
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Calculates the proof-of-work for the current block.
     *
     * @return the hash value of the block after successful proof-of-work
     */
    public String proofOfWork() {
        String hash_value;

        while (true) {
            // Calculate the hash value
            hash_value = calculateHash();
            // Check if the hash value satisfies the difficulty
            if (hash_value.substring(0, difficulty).matches("0{" + difficulty
+ "}")) {
                break;
            }
            // Increment the nonce value and try again
            nonce = nonce.add(BigInteger.ONE);
        }

        return hash_value;
    }

    /**
     * Sets the data of the block.
```

```
     *
     * @param data the data to be set
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Sets the difficulty level for proof-of-work.
     *
     * @param difficulty the difficulty level to be set
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Sets the index of the block.
     *
     * @param index the index to be set
     */
    public void setIndex(int index) {
        this.index = index;
    }

    /**
     * Sets the previous hash of the block.
     *
     * @param previousHash the previous hash to be set
     */
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Sets the timestamp of the block.
     *
     * @param timestamp the timestamp to be set
     */
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

    /**
     * Converts a byte array to a hexadecimal string.
     *
     * @param bytes the byte array to be converted
     * @return the hexadecimal string representation of the byte array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];

        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 255;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 15];
        }
```

```
        return new String(hexChars);
    }
}
```

Project3 Task 2 Exploring Web3 using Algorand

```
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/C3XPY2PUVVUDFIBRET344VNM2YDQRVGRV4ZY22UCFWDTV6LTVSCQ

HTTP/1.1 200 OK
server: nginx
date: Sun, 19 Mar 2023 22:01:01 GMT
content-type: application/json; charset=UTF-8
content-length: 728
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private

{
  "current-round": 28527727,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28527506,
    "fee": 1000,
    "first-valid": 28527504,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "C3XPY2PUVVUDFIBRET344VNM2YDQRVGRV4ZY22UCFWDTV6LTVSCQ",
    "intra-round-offset": 9,
    "last-valid": 28528504,
    "payment-transaction": {
      "amount": 10000000,
      "close-amount": 0,
      "receiver": "FAN6KLIOYHE7B32SD7AF4Q5ZT2ZRJFFXQVWTKJWSUFVBUJDPVRGAJA37BI"
    },
    "receiver-rewards": 0,
    "round-time": 1679262453,
    "sender": "DISPE57MNLYKOMOK3H5IMBAYOYW3YL2CSI6MDOG3RDXSMET35DG4W6SOTI",
    "sender-rewards": 0,
    "signature": {
      "sig": "TPLWVzvI251AMhvGRR7aIqnhHWT+6fsOegpQgGqAwdD0bBJuEaXTRRLmSFx9NwXI9VyRccGGh8053FNXT/MVDQ=="
    },
    "tx-type": "pay"
  }
}

Response file saved.
> 2023-03-19T180101.200.json

Response code: 200 (OK); Time: 557ms (557 ms); Content length: 728 bytes (728 B)
```

```
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/QHD3VSGDTK5G7RXRT6R6XR77FYC2WEOID6PLWUQRJDIG4IA2SICA

HTTP/1.1 200 OK
server: nginx
date: Sun, 19 Mar 2023 22:00:48 GMT
content-type: application/json; charset=UTF-8
content-length: 727
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private

{
  "current-round": 28527724,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28527689,
    "fee": 1000,
    "first-valid": 28527687,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "QHD3VSGDTK5G7RXRT6R6XR77FYC2WEOID6PLWUQRJDIG4IA2SICA",
    "intra-round-offset": 0,
    "last-valid": 28528687,
    "payment-transaction": {
      "amount": 5000000,
      "close-amount": 0,
      "receiver": "K2EP3LIPR3KEI7QOVW3UHLN6JGASMF442YRI5IPO6N6UWPUVNZJ6BVFT4U"
    },
    "receiver-rewards": 0,
    "round-time": 1679263116,
    "sender": "FAN6KLIOYHE7B32SD7AF4Q5ZT2ZRJFFXQVWTKJWSUFVBUJDPVRGAJA37BI",
    "sender-rewards": 0,
    "signature": {
      "sig": "qLmlAZfkUpJ3/xy8yxBwMz6vm2Dmq96I8Amf7KS80ry9+6jKe63p5JlFafKCVAVM0NFNjsTM5SLuq0qNcOt8Dg=="
    },
    "tx-type": "pay"
  }
}

Response file saved.
> 2023-03-19T180048.200.json

Response code: 200 (OK); Time: 846ms (846 ms); Content length: 727 bytes (727 B)
```